

Learning and Adaptivity in Interactive Recommender Systems

Francesco Ricci and Tariq Mahmood¹

Abstract. Recommender systems are intelligent applications that assist the users in a decision-making process by giving personalized product recommendations. Quite recently conversational approaches have been introduced to support a more interactive recommendation process. Notwithstanding the increased interactivity offered by these approaches, the system activity is rigid and follows an execution path that must be defined apriori, at design time. In this paper, we present a new type of recommender system capable of learning an interaction strategy by observing the effects of its actions on the user and on the final outcome of the recommendation session. We view the recommendation process as a sequential decision problem, and we model it as a Markov Decision Process. We evaluate this approach in a case study where the goal is to learn the best support strategy for advising a user in refining a query to a product catalogue. The experimental results demonstrate the value of our approach and show that an initial fixed strategy can be improved by first learning a model of the user behavior and then applying the Policy Iteration algorithm to compute the optimal recommendation policy.

1 INTRODUCTION

Recommender systems are intelligent applications aimed at assisting users in a decision-making process when the user doesn't have sufficient personal experience to choose one item amongst a potentially overwhelming set of alternative products or services [11]. They have been exploited for recommending books, CDs, movies, travels, financial services, and in many other applications [1, 5, 15, 9]. Many recommender systems are designed to support a simple human-computer interaction where two phases can be identified: user model construction and recommendation generation. The user model is typically acquired by either exploiting a collection of previous user-system interactions or information provided by the user during the recommendation session. Then, the recommendation generation reduces to the identification of a subset of products that "match" the user model. For instance, in collaborative filtering the user model is comprised of the ratings provided by the user to a set of products and recommendations are computed by identifying a set of similar users according to the user profiles, and then recommending products highly rated by similar users [14].

This behavior deviates from a more natural human-to-human interaction, where the user and the recommender (advisor) would interact by exchanging requests and replies until the user accepts a recommendation. To this end, *conversational* recommender systems have been proposed. In these systems, there is no clear separation between the user model construction and the recommendation generation stages. In conversational systems a dialogue is supported, where

at each stage the system has many alternative moves; it can ask the user for preferences, or request a feedback on a product or suggest some products [19, 2, 10, 3]. Although such systems bring human-machine interaction a step closer to human-human interaction, they still lack some important features. We believe that a conversational system should be able to generate and update the conversation strategy by exploiting the user model, or beliefs about that, and by considering the feedbacks obtained in previous interactions with the users, e.g., if the recommendation session was successful or not (e.g. something was sold at the end). For instance, imagine that a conversational travel recommender system should suggest a few hotels in a city that would suit the user preferences. The system may adopt different recommendation strategies: it can start querying the user about preferred features and can get enough information to be able to select a small candidate set, or can propose some candidates and acquire user preferences as feedback (critiques) on the illustrated products, or can list products and let the user sort them according to his preferred criteria. Traditional conversational systems hard-code the process and follow it quite rigidly. So, for instance a critique-based recommender will always let the user to make an initial query and then will ask the user for critiques on the top selected products until the user selects a product. In fact, current critique-based recommender can select in an adaptive way the products to recommend (e.g. more similar to the query or more diverse [10]) but cannot decide autonomously, for instance, to stop collecting critiques and offering the user the possibility to sort products according to a criteria. In other words, current conversational systems cannot learn from the experiences collected during the recommendation process and cannot identify the best conversation strategy for generating and offering useful recommendations.

We would like to tackle these requirements by proposing a new type of recommender system that, rather than simply recommending products by following a rigid interaction design, offers a range of information and decision support functions and, during the user-system interaction process, autonomously decides what action to take in order to assist the user in achieving his goals. In this context, the system actions could be very different moves, such as making a recommendation, offering some product information or suggesting to use a product comparison tool. Hence, in our proposed scenario, we assume that at each stage of the process, the user can take some actions, i.e., call some system functionality, such as browse a products' list or query the system for products. Then, the job of the system is to decide what to do, i.e., what computation really to perform and what to show to the user at the next step. The goal of the system is to choose the action that will bring the conversation in the best possible next state, i.e., in a state that is liked by the user and is closer to the goal.

¹ ITC-irst, Italy, email: {ricci, tariq}@itc.it

In the rest of this paper we shall first define more precisely the proposed recommendation model. In Section 2 we shall view the recommendation process as a sequential decision process and we shall introduce the Markov Decision Process (MDP) framework [18]. We define the concept of recommendation policy and we shall explain in which sense a policy can be optimal. Then in Section 3 we shall apply the general model to analyze and optimize a particular type of interaction that occurs quite often in recommender systems, i.e., the incremental request of product features to further constrain a query to a product catalogue. Here the system should basically decide whether to actively ask the user for additional product features, in order to retrieve a small set of candidate products, or let the user to autonomously take this decision and manually change the query. In Section 4 we show that the recommender system, by observing the user responses to system actions, can improve an initial policy that, in a rigid way, dictates to ask the user for additional features only when the result set size is greater than a given threshold. We show that different policies are learned when the agent estimates in a different way the cost of each interaction, i.e., the dissatisfaction of the user to not have reached the goal, that is in this case the selection of the best product. Finally in Section 6 we shall point out some related approaches, stress the limitations of our work and list some future extensions.

2 THE ADAPTIVE RECOMMENDER MODEL

In our proposed model, the recommender system is comprised of two entities, the Information System (IS) and the Recommendation Agent (RA). Basically, IS is the non-adaptive entity which is accessed by the user to obtain information. Its function is entirely controlled by the user and serves requests like displaying a query form page, or displaying the query results. On the other hand, the Recommendation Agent (RA) is the adaptive entity whose role is to support the user in his decision task by helping him to obtain the right information at the right time. For instance, in a travel agency scenario, a traveller browses a catalogue (IS) to get suggestions/recommendations and the travel agent (RA) helps the user to find suitable products by asking questions or pointing to some catalogue pages or products. The distinction between the Information System and the Recommender Agent is transparent to the user but it helps understanding the proposed wider role of the recommender component. In fact, this definition of recommender system contrasts with the classical idea of recommenders as information filtering tools, i.e., applications that can shade irrelevant products and hence alleviate the information overload problem.

To further explain this recommendation process, we assume that the user might execute a number of actions during his interaction with the system. We label these user actions as the *information functions*, since these are requests for various kinds of information offered by the IS. For instance, the user may request the list of top N destinations, or the list of hotels liked by the people he knows. It is worth noting that the user interacts with the system to achieve some goal, e.g., to buy some product, and at each stage of the interaction, he decides to call one among the available information function to attain his goal. The job of the Recommendation Agent is to *decide* what to do after this user request. In fact, this decision is not uniquely identified; the agent can, for instance, show the requested information, or decide to ask for some additional information, or to recommend modifying the current request. The agent's ultimate goal is to take those decisions that: a) the user is more likely to accept rather than reject, and b) in the long run, i.e. at the end of the interaction

session, are more likely to bring the user to his goal, whatever this might be. These decisions are called *system actions*.

As the interaction proceeds, the agent should improve or optimize its decisions, hence must learn an optimal recommendation policy, i.e., that dictating the *best* system action for each possible state of the interaction. This, brings us to model the recommendation process as a Sequential Decision Problem, to exploit the Markov Decision Process (MDP) framework, and to solve the policy learning problem with Reinforcement Learning techniques.

Reinforcement Learning involves learning by interacting with an environment and from the consequences of actions rather than from explicit teaching [18]. Basically, our recommender system (represented by a decision-making agent) interacts with the user, who is part of its environment. The agent perceives aspects of the environment's *state* and takes what it believes to be the best system action to assist the user; the environment responds through a *reward* which is based on the user's response to this action. The agent exploits this reward to learn to take better (system) actions in the future. Reinforcement Learning techniques guarantee that, as the agent-environment interaction proceeds, the agent would eventually learn to take the best actions for all possible environment states. When this happens, we say that the agent has adopted the *optimal recommendation policy*.

2.1 The Markov model of the recommendation agent

In this section, we shall present a general model of the Sequential Recommendation Problem as a Markov Decision Process (MDP). In the next section we shall illustrate an example by completely specifying all the elements of the model (States, Actions, Transitions, Reward). The MDP model of the recommender agent includes:

1. A set of states S representing the different 'situations' the recommendation agent can observe as it interacts with the user. The state representation could include a number of variables related to: the state of the user (U), the state of the agent (A), and the state of the Interaction (I). For instance, there could be a variable that describes the user experience, the number of times the agent made a recommendation, and the last page visited by the user. Basically, a state $s \in S$ must define what is important to know for the agent to take a good action.
2. A set of possible actions A the system can perform in $s \in S$ and that will produce a transition into a next state $s' \in S$. In general, we assume that the agent's environment is non-deterministic, i.e., after executing an action in s , the agent can transit into many alternative states. In other words, the agent does not know what will be the next state of the interaction after he has performed one of his actions. For instance it may "recommend a product" and this could be either selected or discarded by the user.
3. A transition function $T(s, a, s')$ which gives the probability of making a transition from state s to state s' when the agent performs the action a . This function completely describes the non-deterministic nature of the agent's environment.
4. A reward function $R(s, a)$ assigning a reward value (a real number) to the agent for each action a taken in state s . As we are basically interested in systems that aid the user in his decision process, the reward should reflect the user's acceptability of the action a . However, the agent cannot know exactly this function. Therefore, we shall make as few assumptions as possible about the reward assuming that all the terminal (goal) states (for instance a check out state) will have a positive reward and all the other states will

have a small negative one (cost).

When the agent interacts with the environment (the user) it receives a reward at each transition. Although this interaction will have a finite number of steps, we are not sure about their total count. In these situations, the total reward obtained by the agent during a recommendation session is normally computed with an infinite-horizon discounted model [18]:

$$R_T = E\left(\sum_{t=0}^{\infty} \gamma^t R_t\right)$$

where R_t is the reward obtained at the t -th interaction and γ is the discount parameter, $0 < \gamma \leq 1$. The reward that the agent will get during an interaction is determined by the *policy* of the agent, i.e., by a function $\pi : S \rightarrow A$ that indicates for each state the action taken by the agent in that state. The expected infinite-horizon discounted sum of rewards obtained by the agent using policy π starting from s is called the *value* of policy π in state s , denoted with V^π , and it satisfies the following recursive equation:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s')$$

The optimal behavior of the agent is given by a policy $\pi^* : S \rightarrow A$ such that, for each initial state s , if the agent behaves accordingly to the policy then the expected total reward is maximum. The maximum total reward that the agent can obtain starting from s and using the optimal policy, is the value of state s :

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t R_t\right)$$

The optimal value is unique and is the solution of the equations:

$$V^*(s) = \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s')), \forall s \in S$$

Given the optimal value, the optimal policy is given by:

$$\pi^*(s) = \arg \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s'))$$

If the transition probabilities are known, the optimal policy can be computed with the Policy Iteration algorithm [18]. In this algorithm, given an initial policy π_0 , the value function of that policy, V^{π_0} , is computed and then the policy is improved by assigning to each state s the action a that maximizes the term $[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^{\pi_0}(s')]$. Then the value function for this new policy is computed and the policy is again improved. This process continues until π_i converges to the optimal policy.

3 CASE STUDY

In this section, we will consider the application of the proposed recommendation methodology to the Query Tightening Process QTP supported by NutKing, a conversational recommender system that combine interactive query management with collaborative ranking [8]. In NutKing a user searching for travel products is asked to specify a simple query (conjunction of constraints on the product features) to search products in a catalogue. If the query retrieves either too many or no products, the system suggests useful query changes

that save the gist of the original request and help the user solving the interaction problem. This process goes on iteratively till a reasonable number of products is selected. We are here concerned with the situation where too many products are retrieved, hence the system uses a feature selection method to select three features [8]. If the user accepts one of these suggestions, he is supposed to provide a preferred value for a feature, hence generating an additional constraints that could reduce the size of the retrieval set.

The recommendation (tightening) policy of NutKing always suggests three features when the result set is larger than a threshold (typically set to 50). Otherwise, it executes the query and shows the list of retrieved products. In this study, we wish to determine whether this policy is optimal, i.e., if this suggestion really can save time and would maximize the probability that the user will ultimately find the desired product.

3.1 The MDP model of QTP

We will now describe the MDP model for QTP. Before defining the states, actions, and rewards of this model we must describe the information system and the user actions. In this case the Information System comprises five web-pages $P = \{S, QF, RT, T, G\}$ and six information functions $F = \{go, execq, modq, acct, rej, add\}$, or user actions, that we shall describe soon. At each stage of the user-system interaction, the user requests a function $f \in F$ in some page $p \in P$. In response, the recommendation agent will take some action, causing a transition into another page p' . This process continues until the interaction terminates. The following figure illustrates the user actions in each page for the QTP model:

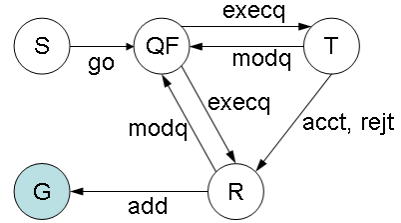


Figure 1. User Interaction Graph

In the start page (S), which is shown to the user each time he logs on to the system the user can only decide to continue by selecting the "go" function. This always causes a transition to the page Query Form (QF) in which the user formulates, modifies and executes his queries. If in QF the user executes a query (execq) the system can transit to either the tightening page (T) or the result set (R). The outcome depends on the system action that will be performed when the user will request to execute a query. Please note that MDP describes the actions of the system and not the actions of the user. The actions of the user and the page where the action is performed are considered as the state definition (see below the definition of states). In the tightening page (T) the user can either accept one of the three proposed features for tightening (acct) or reject the suggestion (rej). Both of these user actions will produce a result page (R), but with different results, in fact if the user rejected the tightening suggestion, then the original query is executed, whether if the user accepted the tightening and provided a feature value then the user modified query is executed by the system. Finally, in the result set page (R) the user can either add a product to the cart (add) or go back to the query form (QF), requesting to modify the current query (modq).

3.1.1 State Space Model

In QTP the state model is described by the following variables:

1. The **Page-UserAction variable** pua , which lists the possible combinations of pages $p \in P = \{S, QF, RT, T, G\}$ and user actions $ua \in UA = \{go, execq, modq, acct, rejt, add\}$. Thus, from Figure 1, the set of these combinations is $PUA = \{S - go, QF - execq, T - acct, T - rejt, T - modq, R - modq, R - add, G\}$.
2. The **result set size c of the current query** which can take on of the 3 qualitative values *small*, *medium*, *large*. A small value is that of a result set of size smaller than c_1 . A medium value is between c_1 and c_2 and a large value is above c_2 . In the experiments we set $c_1 = 20$ and $c_2 = 40$. Furthermore, we assume that $c = large$ in the initial state with $pua = S - go$ because we assume that the initial query is the empty one.
3. The **estimated result set size ec after tightening** is the (qualitative) system estimation of the result set size after the best feature is used for tightening by the user. (See Section 4.1). We assume that $ec \in \{small, medium, large\}$ and that $ec = large$ in the initial state as above for the c variable.

As $|PUA| = 8$ and both c and ec can take three values, the number of possible states is $8 * 3 * 3 = 63$. However, as ec can never be greater than c and there is only one (initial) state with $pua = s - go$ which we consider separately, there are 6 (c, ec) possible combinations and the total number of states is $7 * 6 + 1 = 43$. These combinations are: $(s, s), (m, s), (m, m), (l, s), (l, m), (l, l)$, where $s = small, m = medium$ and $l = large$.

3.1.2 System actions and state transition diagram

In QTP there are four possible system actions: show the query form page (*showQF*); suggest tightening features (*sugg*); execute the current query (*exec*); add a product to cart (*add*). The following diagram depicts how these actions (black nodes) cause transitions between the states (white nodes). For simplicity, we illustrate below only the pua variable in the state representation.

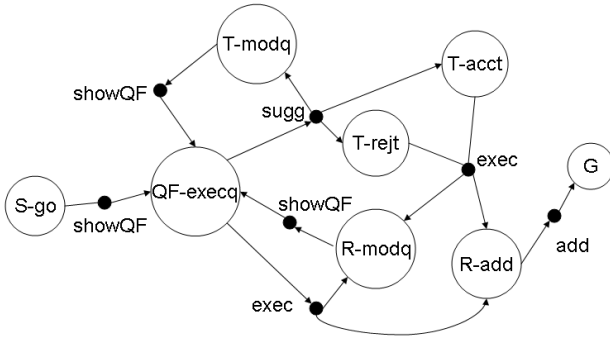


Figure 2. State Transition Diagram

In state $S - go$, the agent executes *showQF* that causes a transition into state $QF - execq$. This state models a situation where the query form is displayed, the user has defined a query and has requested its execution. From here, according to the values of c and ec , the system decides to execute the query (*exec*) going either to states $R - modq$ or $R - add$, or to suggest tightening (*sugg*). In the first

case the outcome depends on the user; if she decides to modify the query ($R - modq$) or to add an item to the cart ($R - add$). If the system suggests features for tightening (*sugg*) then the transition could be to state: $T - acct$, if the user accept tightening; $T - rejt$ if the user reject tightening, or $T - modq$ if the user decides to modify the query. In the states $T - acct$ and $T - rejt$ the system can only execute either the tightened query or the original query, respectively. In states $T - modq$ and $R - modq$ the only possible action for the agent is to show the query form *showQF* to allow the user to modify the query. Finally, in state $R - add$, the agent can only proceed to G (*add*).

We observe from Figure 2 that the MDP system has a stochastic behavior only for a small subset (4) of the possible state-actions combinations. In fact, only the state-actions: ($QF - execq, sugg$), ($QF - execq, exec$), ($T - acct, exec$), and ($T - rejt, exec$), the outcome of the system action depends on the user choice.

3.1.3 Reward

The reward for a system action is defined in such a way that: for each transition into a non-terminal state, the agent is penalized with a negative reward *cost*, unless it transits to the goal state, where the agent gets a +1 reward. In the experiments below we shall see how different *cost* values will impact on the optimal strategy. We recall that a negative cost for each non terminal transition models the fact that each interaction has a "cognitive" cost, that here we assume to be constant for sake of simplicity.

4 EVALUATION

In this section we shall perform some experiments aimed at showing that the recommendation agent is able to improve an initial recommendation policy as it interacts with the user, and can finally adopt an optimal one. To this end we will conduct some simulations. In these simulations, after having defined a user behavior model, we shall learn the transition probabilities. The user behavior model describes how the user will react to the system actions, hence for instance, it describes when the user will accept or reject a tightening suggestion. After having learned the transition model $T(s, a, s')$ we shall apply the Policy Iteration algorithm [18] to improve the initial policy, i.e., that used by the Nutking Recommender, and to learn the optimal policy.

The evaluation procedure basically simulates user-system sessions or *trials*, where each trial simulates a user incrementally modifying a query to finally select/add a product and is composed of some interactions, with the initial state being $(pua = S - go, c = large, ec = large)$. We run this procedure for a set of randomly selected products (from the catalogue). In each interaction, the user takes some action which is followed by the agent's response (system action). The interactions continue until the goal state is reached ($pua = G$). In this simulation we perform a leave-one-in selection, i.e., for each trial we select a product $t = (v_1, \dots, v_n)$, **test item**, and we simulate a user searching for that item. In the simulation, the values used by the simulated user to tighten a query when a suggested feature is accepted or when a query is modified are those of the test product. Note that not all the features have a value, i.e., some of these v_i may be NULL.

4.1 User behavior model

The user behavior model must tell how the simulated user will behave during the simulated interaction in the following three cases:

- (Case 1) When the user is in state QF-execq, how she will modify the current query.
- (Case 2) When the system suggests a tightening (sugg), if the user will decide to modify the query (T-modq) or accept the tightening (T-acct) or reject the tightening (T-rejt).
- (Case 3) When the system execute a query (exec), if the user will add the test item to the cart (R-add) or modify the query (R-modq).

Let us now describe these three situations. (Case 1) At the beginning of each session, we sort the features of the test product according to their frequency of usage (as observed in real interactions with NutKing [12]). Then we use this sorting to choose the first feature to use in the initial query and to incrementally select the next feature to constrain when the user is back to state QF-execq. Suppose that for instance the first 4 sorted features are (f_1, f_5, f_4, f_3) , then the first query is $q = (f_1 = v_1)$ (where v_1 is the value of the first feature in the test item). Then, if the user decides to modify q , the next constraint that will be added is $(f_5 = v_5)$ and the query will become $q = (f_1 = v_1) \text{AND} (f_5 = v_5)$.

When system suggests a tightening (Case 2), we may have three outcomes:

1. The user accepts tightening (T-acct) if one of the three suggested features has a non NULL value in the current test item. If this holds the first of these feature, in the preference order of the user, having non NULL value, is used to further constrain the query, using the value specified in the test item (as above in Case 1).
2. If the user doesn't accept tightening, and the result set size is smaller than a threshold (40 in the experiments) then the user rejects it and executes the original query (R-rejt).
3. In the remaining cases (i.e., for large result sets and when the tightening suggestion does not indicate a feature for whom the user is supposed to have a preferred value) the user is supposed to modify autonomously the query (T-modq), as described for Case 1.

In Case 3, the user will add the test item to the cart (R-add) if the test item is found in the top N items returned by the query (N=3 in the experiments), otherwise the user is supposed to opt for a modification of the current query (R-modq).

4.2 Model learning and optimal policy

Before applying the Policy Iteration algorithm, we shall first use the user behavior model to learn the transition probabilities. The process of learning the transition model is a supervised learning task where the input is a state-action pair and the output is the resulting state. We keep track of how often each action outcome occurs and make a maximum likelihood estimate of the probability $T(s, a, s')$ using the frequency of reaching s' when a is executed in s . To this end, we run the Passive ADP algorithm [13] for 6000 trials, which captures the experience generated with a fixed policy to update the transition probabilities. A fixed policy implies that the agent doesn't explore its environment and its actions in each state remain fixed. However, the success of an optimal policy depends heavily on exploration while gaining experience [20]. In our (simple) model, the agent must really choose one action among two alternatives (sugg or exec) only in the six states with $pua = QF - execq$. As there are two possible actions, the system can adopt a total of $2^6 = 64$ possible policies. We generate exploratory behavior by alternating the fixed policy amongst these 64 by randomly selecting an action for each of the 6 states after every 150 trials. Hence, for the initial evaluation, we consider experience generated with only $6000 \div 150 = 40$ of the 64 policies, not all of them being unique.

At this step, as we mentioned above, we apply the Policy Iteration algorithm to learn the optimal policy. Here we will determine how the optimal policy is influenced by changes in the MDP model. More specifically, we want to analyze the policy improvement process, and the resulting optimal policy for different negative rewards (*cost*) for all the transitions into the non-terminal states (G). These transitions are the intermediate steps in the recommendation process, and *cost* models the user's (cognitive) cost in passing through an additional step of the recommendation interaction.

5 RESULTS AND DISCUSSION

In our experiments we considered five possible values for *cost*: -0.01,-0.02,-0.04,-0.08,-0.12. Moreover we set the discount rate $\gamma = 0.85$, allowing the policy evaluation phase of the Policy Iteration algorithm to run for 100 trials and the algorithm itself to run for 10 policy improvement iterations. We shall show here only the policy behavior in the states with $pua = QF - execq$, i.e., when the agent must select whether to suggest tightening or execute the query. Thus, we will list the prescribed actions (under the optimal policy) for only 6 states, i.e., for all the possible combinations of the remaining state variables, c and ec . We recall that in the initial policy (implemented in NutKing), the agent suggests tightening only when $c = large$; otherwise it executes the query. Table 1 shows the optimal policy actions for the above mentioned 6 states obtained under varying *cost* values, and the initial policy actions (Init Pol. row), that are not optimal. Here, the 6 combinations of variables c and ec are shown in abbreviated form. Hence, for instance the column (l, s) shows the action prescribed for different *cost* values when $c = large$ and $ec = small$.

Optimal Policy Actions for states with $pua = qf - eq$						
<i>cost</i>	(s, s)	(m, s)	(m, m)	(l, s)	(l, m)	(l, l)
<i>Init Pol.</i>	exec	exec	exec	sugg	sugg	sugg
-0.01	exec	exec	exec	exec	exec	exec
-0.02	exec	exec	exec	exec	exec	sugg
-0.04	exec	exec	sugg	exec	sugg	sugg
-0.08	exec	sugg	sugg	sugg	sugg	sugg
-0.12	sugg	sugg	sugg	sugg	sugg	sugg

s : small m : medium l : large $sugg$: suggest $exec$: execute

Table 1. Optimal Policies for Varying Interaction Costs.

These results illustrate that for each value of *cost*, the agent is able to improve an initial policy and to finally adopt an optimal one, and the policy depends on the value of the interaction cost. We observe that when the cost is small (-0.01), the agent has learned that executing the query for all the states is the optimal policy. On the other extreme, when the cost is large (-0.12), the optimal policy suggests tightening for all the states.

In order to understand these results, let us consider a situation where the user has formulated a query q that has a result set that doesn't contain the current test product in the top three positions; hence, the goal state cannot be reached. Let us further assume that there is a feature that if constrained in q will make the result set smaller and the test product to be in one of the top three positions; hence, the goal state will be reached.

Referring to Figure 2, if the system is in state $QF - execq$ and the agent executes q (action *exec*), then the state $R - add$ will be

reached after 3 transitions: first moving to $R - modq$, then again to $QF - execq$ (since the target product is not in the first 3 positions), then finally, after a manual modification of the query to $R - add$, since we are assuming that, with one additional feature constrained in it, the query will return a result set with the test product in the top three positions. Conversely, if the system suggests tightening, the user can accept it with some probability, which can be either large or small. In this case, with only 2 transitions the system will get to state $R - add$ since in the state $T - acct$ the query is modified and the new feature is added to the query. Hence, we see that when the interaction cost is high the system will try actions that have a certain probability to reduce the interaction length, even if this probability is small. On the other hand, when the cost is low the system will tend to execute actions that increase the interaction length but guarantee that the goal state would be finally reached. The behavior of the optimal policies obtained for the remaining values of *cost* varies between the two described above. As the interaction cost increases, the system prefers to suggest tightening for more and more states.

6 CONCLUSIONS

The application of the MDP model to dialog management for learning an optimal dialog strategy has already achieved promising successes in similar (yet simpler) tasks [7, 17]. In these applications the dialogue is aimed at acquiring specific values from the user, the states are all the possible dialogues and the system actions are possible speech-synthesized utterances.

The research work in the domain of MDP-based recommender systems is still in its infancy, with only limited results and with a quite different model of the overall recommendation process. In [16] the authors model the states as the set of products previously bought by a user and the recommender actions correspond to the next possible recommendations. Hence, in this case, the goal is similar to that of a classical recommender system, i.e., to learn what products to recommend next rather than to decide what action to choose in a more diverse set of possible moves. Reinforcement Learning in adaptive recommender systems has also been used in [6] to identify the best recommendation algorithm among some competitive alternatives. Our approach is strongly inspired by [4] where the objective is to design a system that actively monitors a user attempting a task and offers assistance in the form of task guidance. In this application the process describes a handwashing task and the system is supposed to provide cognitive assistance to people with Alzheimer's disease.

In conclusion, in this paper, we have provided a new application of MDP techniques to recommender systems and we have shown that more adaptive recommendation processes can be supported. The experimental study shows promising results in a limited situation. In our future work, we intend to evaluate our approach under a more complex system configuration, for instance, under a larger number of system and user actions, and a larger number of states. We plan to refine the model-learning process by considering more sophisticated techniques as in [20]. Furthermore, we intend to involve real users in our next experiments, to study the effect of policy learning, on the user behavior, and in particular the effect of suboptimal actions in the exploration stages.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin, 'Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *IEEE Transactions on Knowledge and Data Engineering*, **17**(6), 734–749, (2005).

[2] D.W. Aha and L.A. Breslow, 'Refining conversational case libraries', in *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR-97)*, pp. 267–278. Springer, (1997).

[3] L. Ardissono, A. Goy, G. Petrone, A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, and R. Schaefer, 'A framework for the development of personalized, distributed web-based configuration systems', *AI Magazine*, 93–110, (2003).

[4] Jennifer Boger, Pascal Poupart, Jesse Hoey, Craig Boutilier, Geoff Fernie, and Alex Mihailidis, 'A decision-theoretic approach to task assistance for persons with dementia', in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, Scotland, (2005).

[5] Robin Burke, 'Hybrid recommender systems: Survey and experiments', *User Modeling and User-Adapted Interaction*, **12**(4), 331–370, (2002).

[6] Nick Golovin and Erhard Rahm, 'Reinforcement learning architecture for web recommendations.', in *International Conference on Information Technology: Coding and Computing (ITCC'04), Volume 1, April 5-7, 2004, Las Vegas, Nevada, USA*, pp. 398–402, (2004).

[7] Esther Levin, Roberto Pieraccini, and Wieland Eckert, 'A stochastic model of human-machine interaction for learning dialogue strategies', *IEEE Transactions on Speech and Audio Processing*, **8**(1), 11–23, (2000).

[8] Nader Mirzadeh, Francesco Ricci, and Mukesh Bansal, 'Feature selection methods for conversational recommender systems', in *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Services*, Hong Kong, (29 March - 1 April 2005). IEEE Press.

[9] Miquel Montaner, Beatriz López, and Josep Lluís de la Rosa, 'A taxonomy of recommender agents on the internet.', *Artificial Intelligence Review*, **19**(4), 285–330, (2003).

[10] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth, 'Incremental critiquing', *Knowledge-Based Systems*, **18**(4-5), 143–151, (2005).

[11] Paul Resnick and Hal R. Varian, 'Recommender systems', *Communications of the ACM*, **40**(3), 56–58, (1997).

[12] Francesco Ricci, Adriano Venturini, Dario Cavada, Nader Mirzadeh, Dennis Blaas, and Marisa Nones, 'Product recommendation with interactive query management and twofold similarity', in *ICCBR 2003, the 5th International Conference on Case-Based Reasoning*, eds., Agnar Aamodt, Derek Bridge, and Kevin Ashley, pp. 479–493, Trondheim, Norway, (June 23-26 2003).

[13] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, second edn., 2003.

[14] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl, 'Analysis of recommendation algorithms for e-commerce.', in *ACM Conference on Electronic Commerce*, pp. 158–167, (2000).

[15] J. Ben Schafer, Joseph A. Konstan, and John Riedl, 'E-commerce recommendation applications', *Data Mining and Knowledge Discovery*, **5**(1/2), 115–153, (2001).

[16] Guy Shani, David Heckerman, and Ronen I. Brafman, 'An mdp-based recommender system', *Journal of Machine Learning Research*, **6**, 1265–1295, (2005).

[17] Satinder P. Singh, Michael J. Kearns, Diane J. Litman, and Marilyn A. Walker, 'Reinforcement learning for spoken dialogue systems', in *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pp. 956–962, (1999).

[18] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[19] Cynthia A. Thompson, Mehmet H. Goker, and Pat Langley, 'A personalized system for conversational recommendations', *Artificial Intelligence Research*, **21**, 393–428, (2004).

[20] M. A. Wiering and J. Schmidhuber, 'Efficient model-based exploration', in *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 6*, eds., J. A. Meyer and S. W. Wilson, pp. 223–228. MIT Press/Bradford Books, (1998).